# 클라우드 게이밍 시스템 개발

팀명 : **Tree**

정보컴퓨터공학부 202155581 이수빈
정보컴퓨터공학부 202055574 이다은
정보컴퓨터공학부 201914130 이지민

2024년 07월 22일

지도교수 : 김 원 석

# 목차

# 1. 요구 조건 및 제약 사항 분석에 대한 수정 사항

## a. 요구 조건

본 과제는 게임 3D 게임인 FPS Microgame 기반의 게임을 구현하여 클라우드 게이밍 시스템을 개발합니다. 하드웨어에 종속되지 않고 실행할 수 있는 게임 환경을 Unity Render Streaming과 WebRTC를 이용하여 개발합니다. 본 보고서는 WebRTC에 대한 자세한 설명을 작성하였으며 WebRTC가 제공하는 서비스에 대한 작동원리, HLS 및 Unity WebGL의 Latency 발생 차이점, Kubernetes를 활용한 컨테이너 관리에 대한 내용을 작성하였습니다.

## b. 수정 사항

기존 Database를 MySQL에서 PostgreSQL로 변경하였습니다. MySQL과 달리 PostgreSQL은 확장성을 중시하며 복잡한 쿼리 처리가 가능하고 무결성을 강조합니다. 기본적으로 모든 트랜잭션에서 기본적으로 ACID를 완벽하게 지원합니다. 또한 논블로킹 및 비동기식 작업을 처리하는 기능이 뛰어납니다. 따라서 고성능 처리가 가능하고 낮은 Latency를 가지며 효율적인 지원 사용이 가능한 PostgreSQL로 변경을 결정하였습니다.

기존 게임을 뱀파이어 서바이벌 게임 기반의 2D 게임에서 FPS Microgame 기반의 3D 게임으로 변경하였습니다. 기존의 2D 게임에서는 상대적으로 3D 게임보다 컴퓨터 리소스 사용량이 적습니다. 3D 게임에서는 3D 그래픽의 복잡성과 이를 처리하기 위한 추가적인 계산 작업 때문에 더욱 많은 컴퓨터 리소스를 요구합니다. 네이티브 환경과 클라우드 스트리밍 환경에서의 리소스 사용량을 비교할 때 좀 더 유의미한 차이를 보일 것으로 예상됩니다. 따라서 게임을 2D게임에서 3D 게임으로 변경하였습니다.

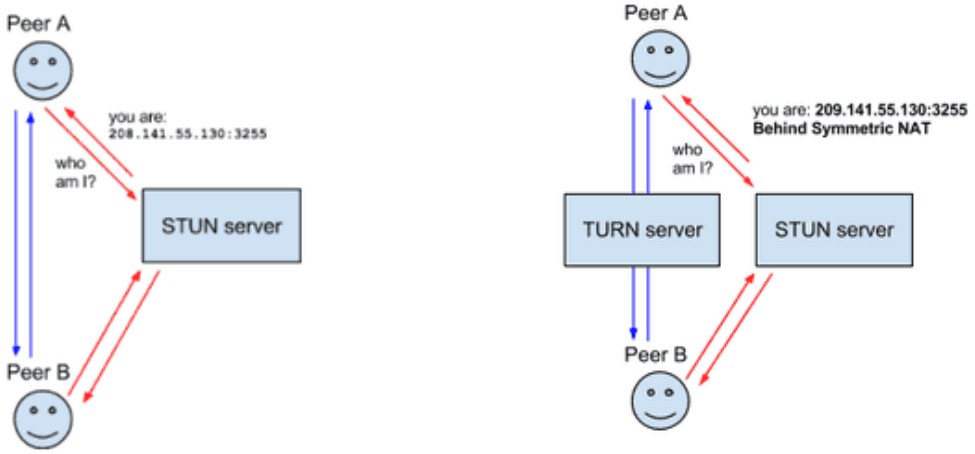
# 2. 설계 상세화 및 변경 내역

## a. WebRTC

- WebRTC (Web Real-Time Communication)는 웹 애플리케이션과 사이트가 중간자 없이 브라우저 간에 오디오나 영상 미디어를 스트리밍하고 임의의 데이터도 교환할 수 있도록 하는 기술입니다. 즉, WebRTC를 이용하면 브라우저 간의 P2P 연결을 통하여 실시간 데이터 교환을 가능하게 할 수 있습니다. 저희 프로젝트에서는 WebRTC를 이용하여 unity server에서 브라우저로의 영상 스트리밍 및 브라우저에서 unity server로의 마우스 키보드 입력 데이터 교환할 것입니다.

- STUN(Session Traversal Utilities for NAT) 서버
  Unity Server와 브라우저간의 P2P 연결을 위해서는 브라우저의 공인 ip 및 port 번호를 알아야합니다. 하지만, 일반적인 브라우저는 웹 서버가 아니기 때문에, 외부에서 접근가능할 수 있는 주소가 없습니다. 이를 라우터를 통과해서 개인의 공인 ip주소와 port 번호를 찾는 과정인 NAT traversal을 이용하여 해결할 수 있습니다. 그리하여 WebRTC는 P2P연결 기반이지만 초기 통신 설정을 위해 NAT traversal 작업을 수행하는 STUN 서버를 활용합니다. 이를 통해 두 단말은 각각 현재 연결된 라우터의 공인 IP 주소와 포트번호를 알게 되어 직접 통신할 수 있습니다.

- TURN(Traversal Using Relay NAT) 서버
  STUN 서버를 이용하더라도, 라우터의 방화벽 정책 혹은 보안 사항으로 인해 공인 ip 주소와 port 번호를 할당받지 못 할 수도 있습니다. STUN 서버로 부터 정보를 찾지 못하는 경우를 대비하기 위해 TURN 서버를 활용합니다. TURN 서버는 네트워크 미디어 패킷을 중개하여 릴레이(relay) 방식으로 통신을 유지합니다.




<그림 1. WebRTC 서버 구성 방식 (왼쪽) STUN 서버, (오른쪽) TURN 서버>

## b. Unity

- 네이티브 환경에서 게임을 실행했을 때와 클라우드 스트리밍 환경에서 실행했을 때의 컴퓨팅 리소스 사용량 차이를 알아보기 위해 비교적 고사양인 3D 게임의 FPS Microgame을 이용합니다.
- Unity Render Streaming API
  사용자의 브라우저와 P2P 연결을 하여 실시간으로 상호작용하기 위하여, Unity에서 제공해주는 실시간 3D 컨텐츠를 스트리밍 하는 기술인 Unity Render Streaming API를 활용합니다.
  Unity Render Streaming API는 실시간 렌더링 및 스트리밍을 지원하여 Unity 애플리케이션의 화면을 실시간으로 캡쳐하여 스트리밍 할 수 있습니다. 단순히 게임 화면을 스트리밍하는 것 뿐만 아니라 사용자들의 마우스나 키보드 입력 등 양방향 인터랙션이 가능합니다.

## c. Backend & Database

- MSA 구조를 기반으로 한 데이터베이스 설계
    - Player Service : player_db (Table : player)
    - Inventory  Service : inventory_db (Table : inventory)
    - Item Service : item_db (Table : item)
    - Stage Service : stage_db (Table : stage)
    - Battle Service : battle_db (Table : wave, wave_monster)
    - Monster Service : monster_db (Table : monster)
    - Character Service : character_db (Table : character)

각 서비스는 각각의 연관된 데이터베이스를 가지는 구조로 설계하였습니다.

- Spring은 기본적으로 하나의 Database를 인식합니다. 이 문제를 해결하기 위해 yaml 파일 설정을 변경하고 추가적인 Configuration 코드를 작성하여 여러 개의 Database가 연결될 수 있도록 설정하였습니다.

## d. Latency

### 1) HLS

- Apple이 개발한 HTTP 기반의 스트리밍 프로토콜이며 주로 비디오 스트리밍에 사용됩니다.
- 기본적으로 비디오 파일을 청크 단위로 전달합니다. 일반적으로 청크 크기는 2 ~ 10초 정도이며 클라이언트가 이 청크들을 다운로드한 후 재생합니다. 이로 인해 HLS는 구조적으로 지연 시간이 더 길어질 수 밖에 없습니다.
- HLS는 버퍼링을 통해 안정적인 재생을 보장하기 위해 초기 재생 전에도 일정량의 데이터를  다운로드하고 버퍼링합니다. 이로 인해 스트리밍 과정에서 HTTP 연결 지연 및 네트워크 지연이 추가적으로 발생할 수 있습니다.

### 2) Unity WebGL

- Unity 엔진을 사용해 제작된 콘텐츠를 웹 브라우저에서 실행할 수 있도록 해주는 기술입니다. 주로 실시간 렌더링 게임이나 상호작용 애플리케이션에 사용됩니다.
- Unity WebGL에서는 게임이나 애플리케이션이 클라이언트 측에서 거의 실시간으로 실행됩니다. 대부분의  입력과 렌더링이 실시간으로 이루어지므로 지연 시간이 매우 짧습니다.

- WebGL 애플리케이션은 초기 로드 시 필요한 리소스를 모두 다운로드한 후 클라이언트에서 실행됩니다. 네트워크를 통한 데이터 전송은 주로 초기 로드 시 발생하며, 이후에는 네트워크 지연이 게임 플레이에 미치는 영향이 매우 제한적입니다.
- WebGL은 브라우저 내 GPU를 사용해 그래픽을 처리하기 때문에 렌더링 지연이 매우 낮으며 실시간 상호작용에 최적화되어 있습니다.

| **HLS** | 비교 | **Unity WebGL** |
|---|---|---|
| 주로 비디오 스트리밍에 사용 | 사용처 | 주로 실시간 렌더링 게임에 사용 |
| 청크 단위 전달 | 전달 방식 | 입력과 렌더링이 실시간 |
| HTTP 연결 지연 및 네트워크 지연이 추가적으로 발생 | 데이터 처리 | 브라우저 내 GPU를 사용하기 때문에 렌더링 지연이 낮음 |

<표 1. HLS와 Unity WebGL의 Latency 비교>

## e. Logging

- filebeat로 로그 수집을 한 후 로그 데이터를 logstash 서버로 전송하여 인덱싱을 진행합니다.
- Elastic search와 Kibana를 통해 로깅 시스템을 구축합니다.

## f. Infra

### 1) Horizontal Pod Autoscaler(HPA)를 활용한 애플리케이션 부하 관리

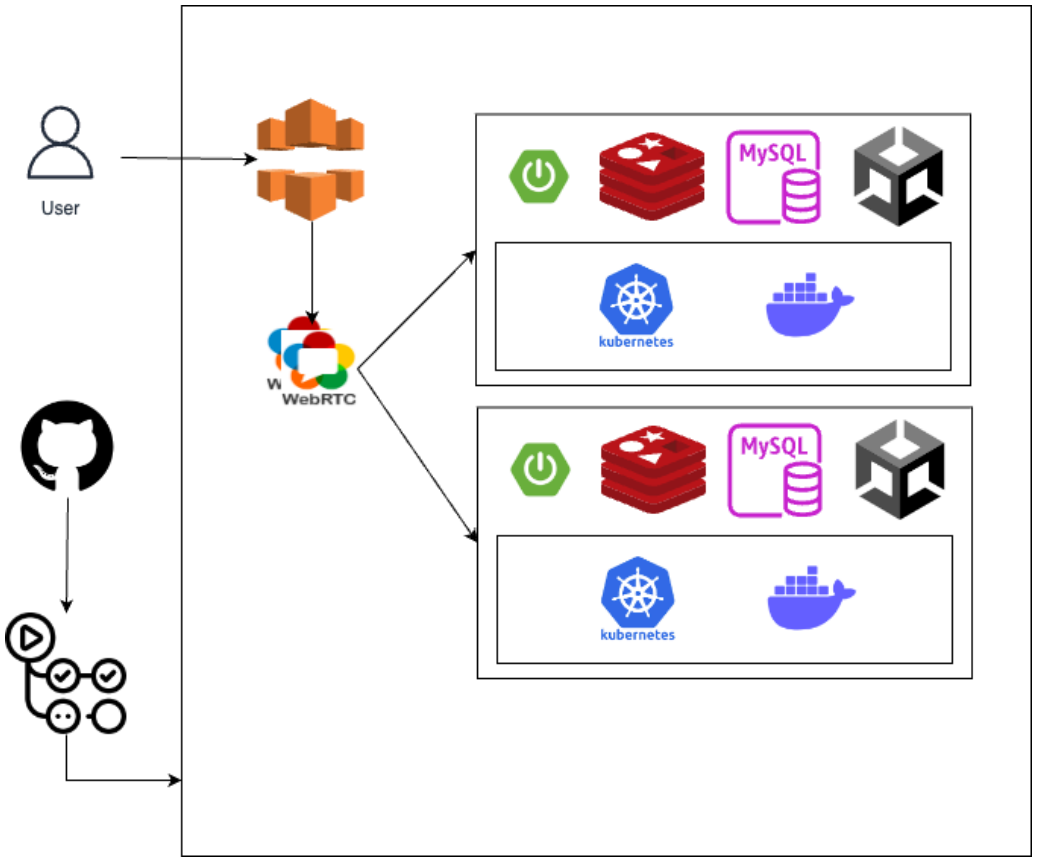
- Horizontal Pod Autoscaler(HPA)는 쿠버네티스에서 제공하는 기능으로, 애플리케이션의 부하를 모니터링하고 설정된 목표(예: CPU 사용률, 메모리 사용률)에 따라 파드의 수를 자동으로 조정합니다. HPA는 클라우드 환경에서 동적으로 변화하는 트래픽이나 부하를 처리하는 데 매우 유용하며, 이를 통해 애플리케이션의 성능을 최적화하고 자원을 효율적으로 사용할 수 있습니다.

- 쿠버네티스는 기본적으로 CPU와 메모리 사용량과 같은 리소스 메트릭을 제공합니다. 이러한 메트릭은 파드와 노드의 리소스 사용량을 실시간으로 수집하며, HPA와 통합되어 CPU 및 메모리 사용률을 기반으로 자동 스케일링을 수행할 수 있고 이를 통해 애플리케이션의 부하를 효과적으로 관리할 수 있습니다.
- Prometheus는 쿠버네티스 환경에서 널리 사용되는 모니터링 도구로, 애플리케이션의 다양한 메트릭을 수집하고 모니터링할 수 있습니다. 애플리케이션은 Prometheus 엔드포인트를 제공하여 주기적으로 메트릭을 수집하게 됩니다. 이를 통해 CPU와 메모리 외에도 초당 HTTP 요청 수, 데이터베이스 쿼리 시간 등 사용자 정의 메트릭을 활용하여 더욱 정밀한 부하 관리를 할 수 있습니다.

### 2) Rolling Update를 통한 무중단 배포

- Rolling Update는 쿠버네티스에서 애플리케이션의 새로운 버전을 배포할 때, 기존 파드를 점진적으로 교체함으로써 서비스 중단 없이 배포를 완료할 수 있도록 하는 배포 전략입니다. 이 방법을 사용하면 기존 파드를 한꺼번에 종료하지 않고, 새로운 버전의 파드를 순차적으로 배포하여 시스템의 안정성을 유지하면서 사용자에게 끊김 없는 서비스를 제공할 수 있습니다.
  - maxUnavailable을 사용해 업데이트 과정에서 사용할 수 없는 파드의 최대 수를 설정할 수 있고, 이를 통해 서비스 가용성을 유지하면서 안정적인 배포가 가능합니다.
  - maxSurge를 사용해 업데이트 과정에서 동시에 생성할 수 있는 새로운 파드의 최대 수를 설정하여, 새로운 파드가 준비되기 전까지 기존 파드가 종료되지 않도록 할 수 있습니다.

## g. 시스템 구성도와 기술 상세 설명




<그림 2. 시스템 구성도>

- AWS CloudFront
  - AWS CloudFront는 정적 및 동적 컨텐츠를 엣지 로케이션에 캐싱에 반복적인 요청에 대한 원본 서버에 접근하지 않고 빠르게 컨텐츠를 제공할 수 있습니다. 클라우드 컴퓨팅 시스템에서는 회원가입과 방 입장 등 게임을 진행할 수 있도록 해 주는 웹 사이트를 제공하는데 사용됩니다.
- WebRTC
  - WebRTC는 웹 브라우저와 모바일 애플리케이션이 별도의 소프트웨어 없이도 실시간으로 오디오, 비디오, 데이터 통신을 할 수 있게 해주는 기술입니다. 클라우드 컴퓨팅 시스템에서는 브라우저 간의 게임 데이터를 교환하고, 오디오와 비디오 스트리밍을 실시간으로 제공할 때 사용합니다.

- Spring Boot
    - Spring Boot는 Java 기반의 프레임워크인 Spring Framework를 기반으로, 개발자들이 더 쉽게 독립 실행형 및 프로덕션급 애플리케이션을 개발할 수 있도록 설계된 프레임워크입니다. 클라우드 컴퓨팅 시스템에서는 사용자의 회원가입과 방 입장, 게임 데이터 저장 등에 관한 로직을 수행하는데 사용됩니다.
- Redis
    - Redis는 인메모리 데이터 구조 저장소로, 다양한 데이터 구조를 저장하고 관리할 수 있도록 하는 NoSQL 데이터베이스입니다. 클라우드 컴퓨팅 시스템에서는 클라이언트의 토큰을 관리하는데 사용됩니다.
- PostgreSQL
    - PostgreSQL은 객체-관계형 Database 관리 시스템입니다. 클라우드 컴퓨팅 시스템에서는 아이템 저장과 게임 스테이지, 몬스터 등에 관한 데이터를 저장하는데 사용됩니다.
- Unity
    - Unity는 2D 및 3D 비디오 게임, 시뮬레이션, 증강 현실 등 다양한 애플리케이션을 개발할 수 있는 크로스 플랫폼 게임 엔진입니다. 클라우드 컴퓨팅 시스템에서는 ‘뱀파이어 서바이벌'을 모작으로 한 게임의 전반적인 진행을 담당하고 있습니다.
- Kubernetes
    - Kubernetes는 오픈 소스 컨테이너 오케스트레이션 플랫폼으로, 컨테이너화된 애플리케이션의 배포, 관리, 확장 및 운영을 자동화 해 주는 도구입니다. 클라우드 컴퓨팅 시스템에서는 현재 MSA 구조로 Spring Boot, Redis 등 많은 컨테이너가 동작하고 있는데, Kubernetes를 활용해 셀프 힐링과 자동화 배포, 모니터링을 진행하고 있습니다.

## 3. 갱신된 과제 추진 계획

| | 8월 | 9월 | 10월 |
|---|---|---|---|
| 1. 게임 개발 마무리<br>2. Backend 개발 part 1<br>3. CICD 구축<br>4. 컨테이너 구축 | | | |
| 1. Frontend 개발<br>2. Backend 개발 part 2<br>3. 모니터링 및 로깅 시스템 구축<br>4. 플랫폼 성능 측정 및 비교 | | | |
| 1. 최종 테스트 및 최종 보고서 작성<br>2. 결과물 업로드 | | | |

<표 2. 갱신된 일정표>

## 4. 구성원 별 진척도

### 이수빈

- 컨테이너 구축 (k8s) - (진행중)
- 모니터링 구축 (Prometheus, Grafana)

### 이다은

- 게임 개발 (Unity & WebRTC) — (진행중)
- Frontend 개발 (React)

### 이지민

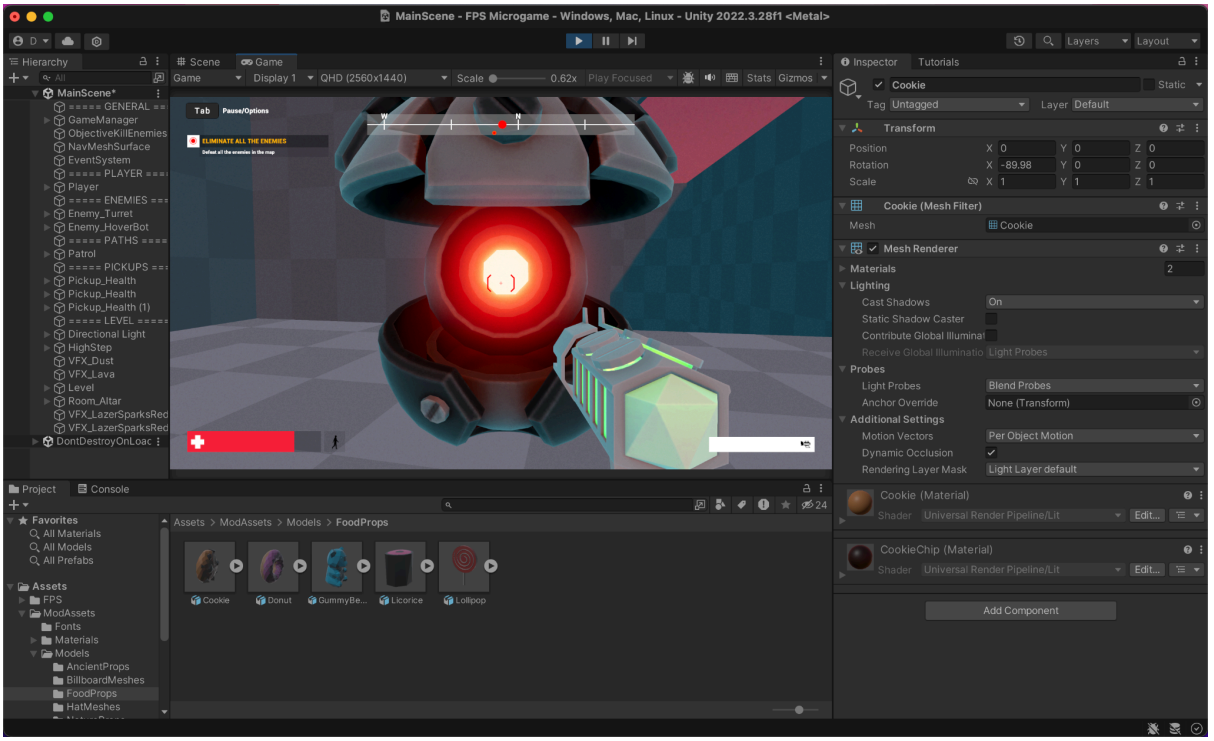
- Backend 개발 (Java & Spring Boot) — (진행중)
- **DB** 구축 **(PostgreSQL & Redis) — (**완료**)**
- CICD 구축 (Github Actions)

### ALL

- **Unity Render Streaming** 기술 바탕 주제 구체화 **— (**완료**)**
- **Architecture** 설계 **— (**완료**)**
- 보고서 작성 **— (**완료**)**

# 5. 보고 시점까지의 과제 수행 내용 및 중간 결과

## a. Unity

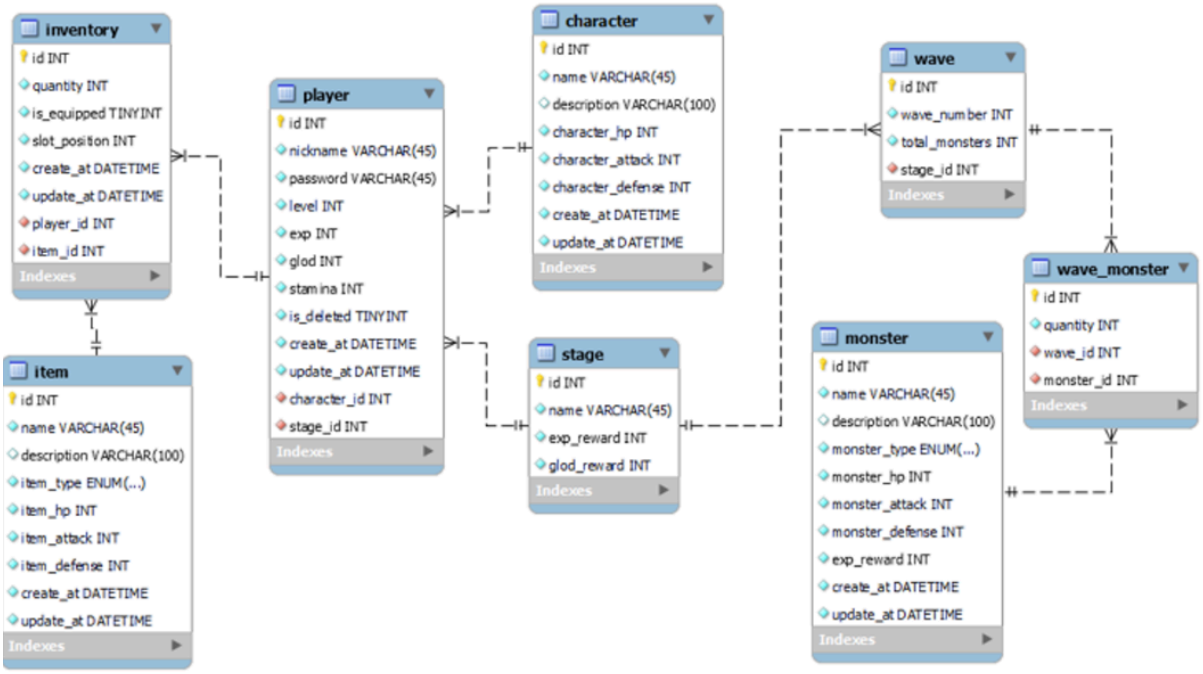



<그림 3. Unity에서 게임을 실행한 모습>

## b. Backend & Database

### 1) MSA 구조를 기반으로 한 데이터베이스 설계

아래의 사진은 실제 데이터베이스의 구조가 아닌 테이블 간의 관계를 시각화하고 이해를 돕기 위해 그린 ERD입니다. 실제 데이터베이스에서는 연관된 테이블 별 각각의 데이터베이스를 가지고 있으며 서비스 API로 동기화를 진행합니다. 또한 Spring에서 각 데이터베이스를 연결합니다.




<그림 4. 데이터베이스의 구조를 ERD로 시각화 >

### 2) Spring과 Database 연동

- 기본적으로 하나의 Database만을 인식할 수 있는 Spring을 추가 설정을 통해 여러개의 Database를 연결할 수 있도록 변경
- Domain 구조 설계

## c. Infra

### 1) Spring boot와 database 연동

```
NAME                        DESIRED   UP-TO-DATE
boot-inventory-service      1         1
boot-item-service           1         1
boot-player-service         1         1
boot-stage-service          1         1
```

<그림 5. Spring boot와 database 연동>

# 6. 참고 문헌


- 조미란, “MSA - 뭉치면 죽고, 흩어지면 산다!", 11번가 주식회사, 2024-08-08, https://techtalk.11stcorp.com/2022/pdf/TECH-TALK-2022_SESSION-15.pdf
- Mdn web docs WebRTC 프로토콜 소개, https://developer.mozilla.org/ko/docs/Web/API/WebRTC_API/Protocols
- 정종윤, “WebRTC는 어떻게 실시간으로 데이터를 교환할 수 있을까?”, 2021-01-24, https://wormwlrm.github.io/2021/01/24/Introducing-WebRTC.html